**The Breakout game**

In this part of the Java 2D games tutorial we will create a simple Breakout game clone.

 The Breakout is an arcade game originally developed by Atari Inc. The game was created in 1976.

In this game, the player moves a paddle on the screen and bounces a ball or balls. The objective is to destroy bricks in the top of the window.

**Development**

In our game, we have one paddle, one ball and 30 bricks. I have created an image for a ball, paddle and a brick in Inkscape. We use a timer to create a game cycle. We do not work with angles, we simply change directions. Top, bottom, left and right. I was inspired by the pybreakout game. It was developed in PyGame library by Nathan Dawson.

The game consists of seven files: Commons.java, Sprite.java, Ball.java, Paddle.java, Brick.java,Board.java, and Breakout.java.

Commons.java

package com.zetcode;

public interface Commons {

public static final int WIDTH = 300;

public static final int HEIGTH = 400;

public static final int BOTTOM\_EDGE = 390;

public static final int N\_OF\_BRICKS = 30;

public static final int INIT\_PADDLE\_X = 200;

public static final int INIT\_PADDLE\_Y = 360;

public static final int INIT\_BALL\_X = 230;

public static final int INIT\_BALL\_Y = 355;

public static final int DELAY = 1000;

public static final int PERIOD = 10;

}

The Commons.java file has some common constants. The WIDTH and HEIGHT constants store the dimensions of the board. When the ball passes the BOTTOM\_EDGE, the game is over. The N\_OF\_BRICKS is the number of bricks in the game. The INIT\_PADDLE\_X and INIT\_PADDLE\_Y are initial coordinates of the paddle object. The INIT\_BALL\_X and INIT\_BALL\_Y are initial coordinates of the ball object. The DELAY is the initial delay in milliseconds before task is to be executed and the PERIOD is the time in milliseconds between successive task executions that form game cycles.

Sprite.java

package com.zetcode;

import java.awt.Image;

import java.awt.Rectangle;

public class Sprite {

protected int x;

protected int y;

protected int i\_width;

protected int i\_heigth;

protected Image image;

public void setX(int x) {

this.x = x;

}

public int getX() {

return x;

}

public void setY(int y) {

this.y = y;

}

public int getY() {

return y;

}

public int getWidth() {

return i\_width;

}

public int getHeight() {

return i\_heigth;

}

Image getImage() {

return image;

}

Rectangle getRect() {

return new Rectangle(x, y,

image.getWidth(null), image.getHeight(null));

}

}

The Sprite class is a base class for all objects in the Board. We put here all methods and variables that are in Ball, Brick, and Paddle objects, like getImage() or getX() methods.

Brick.java

package com.zetcode;

import javax.swing.ImageIcon;

public class Brick extends Sprite {

private boolean destroyed;

public Brick(int x, int y) {

this.x = x;

this.y = y;

ImageIcon ii = new ImageIcon("brick.png");

image = ii.getImage();

i\_width = image.getWidth(null);

i\_heigth = image.getHeight(null);

destroyed = false;

}

public boolean isDestroyed() {

return destroyed;

}

public void setDestroyed(boolean val) {

destroyed = val;

}

}

This is the Brick class.

private boolean destroyed;

In the destroyed variable we keep the state of a brick.

Ball.java

package com.zetcode;

import javax.swing.ImageIcon;

public class Ball extends Sprite implements Commons {

private int xdir;

private int ydir;

public Ball() {

xdir = 1;

ydir = -1;

ImageIcon ii = new ImageIcon("ball.png");

image = ii.getImage();

i\_width = image.getWidth(null);

i\_heigth = image.getHeight(null);

resetState();

}

public void move() {

x += xdir;

y += ydir;

if (x == 0) {

setXDir(1);

}

if (x == WIDTH - i\_width) {

setXDir(-1);

}

if (y == 0) {

setYDir(1);

}

}

private void resetState() {

x = INIT\_BALL\_X;

y = INIT\_BALL\_Y;

}

public void setXDir(int x) {

xdir = x;

}

public void setYDir(int y) {

ydir = y;

}

public int getYDir() {

return ydir;

}

}

This is the Ball class.

public void move() {

x += xdir;

y += ydir;

if (x == 0) {

setXDir(1);

}

if (x == WIDTH - i\_width) {

setXDir(-1);

}

if (y == 0) {

setYDir(1);

}

}

The move() method moves the ball on the Board. If the ball hits the borders, the directions are changed accordingly.

public void setXDir(int x) {

xdir = x;

}

public void setYDir(int y) {

ydir = y;

}

These two methods are called when the ball hits the paddle or a brick.

Paddle.java

package com.zetcode;

import java.awt.event.KeyEvent;

import javax.swing.ImageIcon;

public class Paddle extends Sprite implements Commons {

private int dx;

public Paddle() {

ImageIcon ii = new ImageIcon("paddle.png");

image = ii.getImage();

i\_width = image.getWidth(null);

i\_heigth = image.getHeight(null);

resetState();

}

public void move() {

x += dx;

if (x <= 0) {

x = 0;

}

if (x >= WIDTH - i\_width) {

x = WIDTH - i\_width;

}

}

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();

if (key == KeyEvent.VK\_LEFT) {

dx = -1;

}

if (key == KeyEvent.VK\_RIGHT) {

dx = 1;

}

}

public void keyReleased(KeyEvent e) {

int key = e.getKeyCode();

if (key == KeyEvent.VK\_LEFT) {

dx = 0;

}

if (key == KeyEvent.VK\_RIGHT) {

dx = 0;

}

}

private void resetState() {

x = INIT\_PADDLE\_X;

y = INIT\_PADDLE\_Y;

}

}

This is the Paddle class. It encapsulates the paddle object in the Breakout game. The paddle is controlled with left and right arrow keys. By pressing the arrow key, we set the direction variable. By releasing the arrow key, we set the dx variable to zero. This way the paddle stops moving.

public void move() {

x += dx;

if (x <= 0) {

x = 0;

}

if (x >= WIDTH - i\_width) {

x = WIDTH - i\_width;

}

}

The paddle moves only in the horizontal direction, so we only update the x coordinate. The if conditions ensure that the paddle does not pass the window edges.

Board.java

package com.zetcode;

import java.awt.Color;

import java.awt.Font;

import java.awt.FontMetrics;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Point;

import java.awt.RenderingHints;

import java.awt.Toolkit;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.util.Timer;

import java.util.TimerTask;

import javax.swing.JPanel;

public class Board extends JPanel implements Commons {

private Timer timer;

private String message = "Game Over";

private Ball ball;

private Paddle paddle;

private Brick bricks[];

private boolean ingame = true;

public Board() {

initBoard();

}

private void initBoard() {

addKeyListener(new TAdapter());

setFocusable(true);

bricks = new Brick[N\_OF\_BRICKS];

setDoubleBuffered(true);

timer = new Timer();

timer.scheduleAtFixedRate(new ScheduleTask(), DELAY, PERIOD);

}

@Override

public void addNotify() {

super.addNotify();

gameInit();

}

private void gameInit() {

ball = new Ball();

paddle = new Paddle();

int k = 0;

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 6; j++) {

bricks[k] = new Brick(j \* 40 + 30, i \* 10 + 50);

k++;

}

}

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,

RenderingHints.VALUE\_ANTIALIAS\_ON);

g2d.setRenderingHint(RenderingHints.KEY\_RENDERING,

RenderingHints.VALUE\_RENDER\_QUALITY);

if (ingame) {

drawObjects(g2d);

} else {

gameFinished(g2d);

}

Toolkit.getDefaultToolkit().sync();

}

private void drawObjects(Graphics2D g2d) {

g2d.drawImage(ball.getImage(), ball.getX(), ball.getY(),

ball.getWidth(), ball.getHeight(), this);

g2d.drawImage(paddle.getImage(), paddle.getX(), paddle.getY(),

paddle.getWidth(), paddle.getHeight(), this);

for (int i = 0; i < N\_OF\_BRICKS; i++) {

if (!bricks[i].isDestroyed()) {

g2d.drawImage(bricks[i].getImage(), bricks[i].getX(),

bricks[i].getY(), bricks[i].getWidth(),

bricks[i].getHeight(), this);

}

}

}

private void gameFinished(Graphics2D g2d) {

Font font = new Font("Verdana", Font.BOLD, 18);

FontMetrics metr = this.getFontMetrics(font);

g2d.setColor(Color.BLACK);

g2d.setFont(font);

g2d.drawString(message,

(Commons.WIDTH - metr.stringWidth(message)) / 2,

Commons.WIDTH / 2);

}

private class TAdapter extends KeyAdapter {

@Override

public void keyReleased(KeyEvent e) {

paddle.keyReleased(e);

}

@Override

public void keyPressed(KeyEvent e) {

paddle.keyPressed(e);

}

}

private class ScheduleTask extends TimerTask {

@Override

public void run() {

ball.move();

paddle.move();

checkCollision();

repaint();

}

}

private void stopGame() {

ingame = false;

timer.cancel();

}

private void checkCollision() {

if (ball.getRect().getMaxY() > Commons.BOTTOM\_EDGE) {

stopGame();

}

for (int i = 0, j = 0; i < N\_OF\_BRICKS; i++) {

if (bricks[i].isDestroyed()) {

j++;

}

if (j == N\_OF\_BRICKS) {

message = "Victory";

stopGame();

}

}

if ((ball.getRect()).intersects(paddle.getRect())) {

int paddleLPos = (int) paddle.getRect().getMinX();

int ballLPos = (int) ball.getRect().getMinX();

int first = paddleLPos + 8;

int second = paddleLPos + 16;

int third = paddleLPos + 24;

int fourth = paddleLPos + 32;

if (ballLPos < first) {

ball.setXDir(-1);

ball.setYDir(-1);

}

if (ballLPos >= first && ballLPos < second) {

ball.setXDir(-1);

ball.setYDir(-1 \* ball.getYDir());

}

if (ballLPos >= second && ballLPos < third) {

ball.setXDir(0);

ball.setYDir(-1);

}

if (ballLPos >= third && ballLPos < fourth) {

ball.setXDir(1);

ball.setYDir(-1 \* ball.getYDir());

}

if (ballLPos > fourth) {

ball.setXDir(1);

ball.setYDir(-1);

}

}

for (int i = 0; i < N\_OF\_BRICKS; i++) {

if ((ball.getRect()).intersects(bricks[i].getRect())) {

int ballLeft = (int) ball.getRect().getMinX();

int ballHeight = (int) ball.getRect().getHeight();

int ballWidth = (int) ball.getRect().getWidth();

int ballTop = (int) ball.getRect().getMinY();

Point pointRight = new Point(ballLeft + ballWidth + 1, ballTop);

Point pointLeft = new Point(ballLeft - 1, ballTop);

Point pointTop = new Point(ballLeft, ballTop - 1);

Point pointBottom = new Point(ballLeft, ballTop + ballHeight + 1);

if (!bricks[i].isDestroyed()) {

if (bricks[i].getRect().contains(pointRight)) {

ball.setXDir(-1);

} else if (bricks[i].getRect().contains(pointLeft)) {

ball.setXDir(1);

}

if (bricks[i].getRect().contains(pointTop)) {

ball.setYDir(1);

} else if (bricks[i].getRect().contains(pointBottom)) {

ball.setYDir(-1);

}

bricks[i].setDestroyed(true);

}

}

}

}

}

This is the Board class. Here we put the game logic.

public void gameInit() {

ball = new Ball();

paddle = new Paddle();

int k = 0;

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 6; j++) {

bricks[k] = new Brick(j \* 40 + 30, i \* 10 + 50);

k++;

}

}

}

In the gameInit() method we create a ball, a paddle, and thirty bricks.

if (ingame) {

drawObjects(g2d);

} else {

gameFinished(g2d);

}

Depending on the ingame variable, we either draw all the objects in the drawObjects() method or finish the game with the gameFinished() method.

private void drawObjects(Graphics2D g2d) {

g2d.drawImage(ball.getImage(), ball.getX(), ball.getY(),

ball.getWidth(), ball.getHeight(), this);

g2d.drawImage(paddle.getImage(), paddle.getX(), paddle.getY(),

paddle.getWidth(), paddle.getHeight(), this);

for (int i = 0; i < N\_OF\_BRICKS; i++) {

if (!bricks[i].isDestroyed()) {

g2d.drawImage(bricks[i].getImage(), bricks[i].getX(),

bricks[i].getY(), bricks[i].getWidth(),

bricks[i].getHeight(), this);

}

}

}

The drawObjects() method draws all the objects of the game. The sprites are drawn with thedrawImage() method.

private void gameFinished(Graphics2D g2d) {

Font font = new Font("Verdana", Font.BOLD, 18);

FontMetrics metr = this.getFontMetrics(font);

g2d.setColor(Color.BLACK);

g2d.setFont(font);

g2d.drawString(message,

(Commons.WIDTH - metr.stringWidth(message)) / 2,

Commons.WIDTH / 2);

}

The gameFinished() method draws "Game over" or "Victory" to the middle of the window.

private class ScheduleTask extends TimerTask {

@Override

public void run() {

ball.move();

paddle.move();

checkCollision();

repaint();

}

}

The ScheduleTask is triggerd every PERIOD ms. In its run() method, we move the ball and the paddle. We check for possible collisions and repaint the screen.

if (ball.getRect().getMaxY() > Commons.BOTTOM\_EDGE) {

stopGame();

}

If the ball hits the bottom, we stop the game.

for (int i = 0, j = 0; i < N\_OF\_BRICKS; i++) {

if (bricks[i].isDestroyed()) {

j++;

}

if (j == N\_OF\_BRICKS) {

message = "Victory";

stopGame();

}

}

We check how many bricks are destroyed. If we destroyed all N\_OF\_BRICKS bricks, we win the game.

if (ballLPos < first) {

ball.setXDir(-1);

ball.setYDir(-1);

}

If the ball hits the first part of the paddle, we change the direction of the ball to the north-west.

if (bricks[i].getRect().contains(pointTop)) {

ball.setYDir(1);

}

If the ball hits the bottom of the brick, we change the y direction of the ball; it goes down.

Breakout.java

package com.zetcode;

import java.awt.EventQueue;

import javax.swing.JFrame;

public class Breakout extends JFrame {

public Breakout() {

initUI();

}

private void initUI() {

add(new Board());

setTitle("Breakout");

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setSize(Commons.WIDTH, Commons.HEIGTH);

setLocationRelativeTo(null);

setResizable(false);

setVisible(true);

}

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

Breakout game = new Breakout();

game.setVisible(true);

}

});

}

}

And finally, this is the Breakout class which has the main entry method.

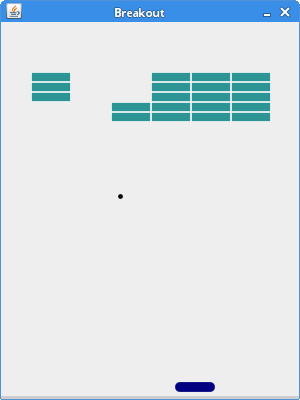


Figure: The Breakout game